VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

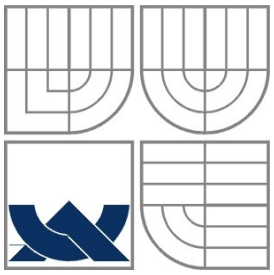
TESTOVÁNÍ PAMĚTI NA ARCHITEKTUŘE SGI/MIPS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

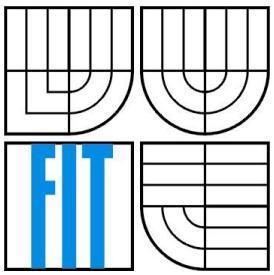
AUTOR PRÁCE
AUTHOR

Karol Rydlo

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TESTOVÁNÍ PAMĚTI NA ARCHITEKTUŘE SGI/MIPS

MEMORY TESTING ON SGI/MIPS ARCHITECTIRE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Karol Rydlo
AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Kašpárek
SUPERVISOR

BRNO 2009

Abstrakt

Moje bakalářská práce se zabývá zprovozněním a vytvořením vlastních testů paměti na grafických stanicích SGI O2, což sebou přináší seznámení se s architekturou procesorů MIPS a pokouší se najít ideální prostředí pro provádění těchto testů. S tím úzce souvisí hledání vhodného způsobu spouštění a překladu aplikací pro stanice SGI O2, kde se zabývá také využitím křížových kompilátorů.

Abstract

Work is engaged in making solution for creating own memory tests on graphical station SGI O2. This thesis produces work on MIPS processor architecture and it try to find the ideal environments for testing memory and with it is nearly related looking for chances of start and compile application for SGI O2. Part of my thesis is also target using cross-compilers, for effective and useful work with program for other architecture.

Klíčová slova

Testování paměti, RAM, SGI, MIPS, O2, Memtest86+, Asembler, SGI PROM, Linux, IRIX, R10000,CRIME, UMA, BOOTP, TFTP, binutils, gcc, křížový kompilátor, Debian, Debian Installer, IP32, ARCLoad, tip22, tip32, jádro, menuconfig

Keywords

Memory testing, RAM, SGI, MIPS, O2, Memtest86+, Assembler, SGI PROM, Linux, IRIX, R10000,CRIME, UMA, BOOTP, TFTP, binutils, gcc, cross-compiler, Debian, Debian Installer, IP32, ARCLoad, tip22, tip32, kernel, menuconfig

Citace

Rydlo Karol: Testování paměti na architektuře SGI/MIPS, bakalářská práce, Brno, FIT VUT v Brně, 2009

Testování paměti na architektuře SGI/MIPS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Tomáše Kašpárka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Karol Rydlo
20.5.2009

Poděkování

Rád bych poděkoval všem, kteří mně pomohli při vytváření mojí bakalářské práce. Především Ing. Tomáši Kašpárkovi za odbornou pomoc a rady kam směřovat moji bakalářskou práci. Dále společnosti Evektor, spol. s r.o. za poskytnutí grafických stanic SGI O2 a také vývojářům linuxové distribuce Debian, Luku Claesovi a Martinu Michlmayrovi, kteří mě byli ochotni dávat potřebné rady a nasměrovat mě tím správným směrem.

Obsah

1 Úvod ... 3
2 Systém SGI O2 ... 4
2.1 Historie ... 4
2.2 Hardware ... 4
2.3 Operační systémy ... 6
2.4 Instrukční sada procesoru R10000 ... 6
2.4.1 MIPS architektura ... 6
2.4.2 Popis procesoru R10000 ... 7
3 Testování paměti ... 8
3.1 Memtest86+ ... 8
3.1.1 Historie Memtestu ... 8
3.1.2 Spuštění ... 8
3.1.3 Instalace memtestu ... 8
3.1.4 Příkazy ... 9
3.1.5 Chyby ... 9
3.1.6 Délka testování paměti ... 10
3.1.7 Princip testování pamětí ... 10
3.1.8 Algoritmus ... 11
3.1.9 Popis jednotlivých testů ... 11
4 Portace Memtestu na architekturu SGI/MIPS ... 14
4.1 Rozdíl architektur ... 14
4.2 Přehled architektury – MIPS asembler ... 14
4.2.1 Datové typy a literály ... 14
4.2.2 Registry ... 14
4.2.3 Programová struktura ... 16
4.2.4 Deklarace dat ... 16
4.2.5 Load / Store Instructions ... 17
4.2.6 Nepřímé a bázové adresování ... 18
4.2.7 Aritmetické instrukce ... 19
4.2.8 Řízení struktury programu ... 19
5 Příprava vývojového prostředí ... 21
5.1 Spuštěni instalátoru ... 21
5.1.1 Instalační medium ... 21
5.1.2 Instalace přes síť ... 21
5.1.3 Zavedení systému ... 23
5.2 Vývojové prostředí na SGI O2 ... 23
5.3 Přechod ze SGI 02 na křížový kompilátor ... 23
5.3.1 Chyby na SGI O2 ... 23

5.3.2 Křížový kompilátor 24
6 Praktická část – vytvoření programu 25
6.1 Test paměti v prostředí PROM 25
6.1.1 ARCLoad 26
6.1.2 Chování aplikace v prostředí PROM 26
6.1.3 Memtester a testování 26
6.2 Test pamětí v jádře Linuxu 27
6.2.1 Vytvoření zavaděče 28
6.2.2 Hledání jádra 31
6.2.3 Úprava jádra 31
6.2.4 Aplikace testující paměť v jádře Linuxu 35
7 Závěr 37
8 Literatura 38
9 Seznam příloh: 39
9.1 CDROM 39

1 Úvod

Grafické stanice od SGI, založené na procesorech architektury MIPS, v sobě spojují několik nových zajímavých témat, především poznání právě jiné architektury než klasické x86, což sebou přináší samozřejmě potřebu vyřešit problémy, které s tím souvisejí, a nalézt použitelné způsoby řešení, abychom mohli spouštět vlastní aplikace, v mém případě program pro testování paměti, na těchto stanicích.

Důvod, proč jsem se rozhodl právě pro testování paměti na této stanici, vychází za prvé z jejího velmi zajímavého a pro mě subjektivně pěkného designu. Samozřejmě design má jen pramálo společného s testováním paměti, ale právě do této oblasti jsem velmi často zabíhal při mých prvních pokusech s těmito stanicemi. A právě fakt, že má první stanice založená na architektuře MIPS, SGI O2, měla těchto problémů více než dost, nebylo již daleko se pustit do vývoje vlastní aplikace pro testování paměti na grafické stanici SGI O2.

Jelikož se však pohybujeme v oblasti architektury MIPS, má teoretická část práce se zaměří právě na popis odlišností a zajímavostí, se kterými se můžeme setkat při práci se stanicemi od SGI, konkrétně SGI O2. V další části se podíváme na principy testování paměti, kde jako vzor pro inspiraci posloužil Memtest86+. S odlišnou architekturou úzce souvisí také jiná instrukční sada, kterou se zabývá třetí část mojí práce, a právě asembler pro MIPS procesory by mohl být tím pravým pro optimalizované algoritmy na testování paměti.

Po teoretickém úvodu zkusíme, zda si dokáže Linux poradit se stanicí SGI O2, zda můžeme právě toto prostředí použít pro tvorbu aplikací nebo zda bude nutné sáhnout např. po křížovém kompilátoru (ang. cross-compiler).

V poslední části mojí práce zavítáme do problematiky jak a kde spustit aplikaci pro testování paměti, hledáním cesty k ideálnímu prostředí pro tyto testy a na závěr si shrneme, jak to všechno dopadlo.

2 Systém SGI O2

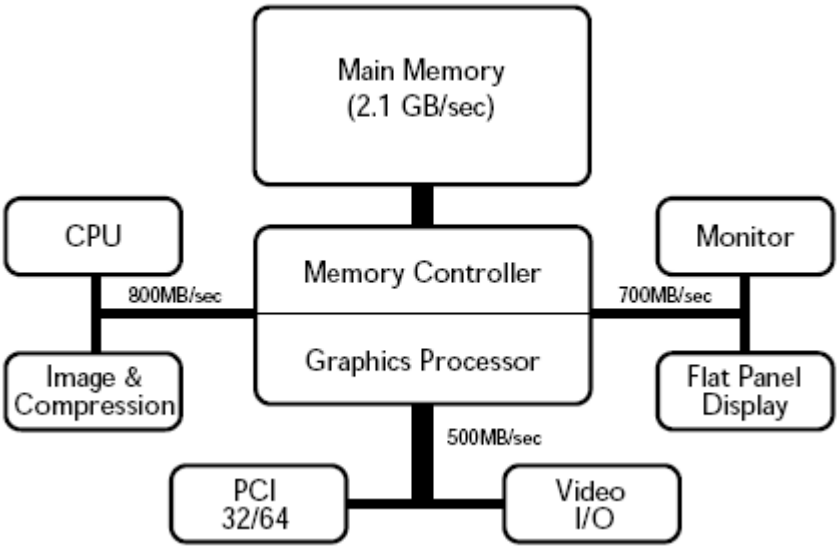
2.1 Historie

Grafická Unixová stanice SGI O2 ("Moosehead") byla společností Silicon Graphics (SGI) uvedena na trh v roce 1996, jakožto nástupkyně předchozí generace především systémů SGI Indy. Základ systému také tvoří jeden MIPS procesor, podobně jako v předchozí generaci, avšak hlavní rozdíl byl především v grafickém subsystému, jelikož grafický čip je na rozdíl od Indyho u O2 přímo součástí základní desky a spolu s dalšími prvky pak tvoří čišet CRIME. Hlavním určením stanice se mělo stát především zpracování multimédií, avšak nakonec své uplatnění nalezla v mnoha oborech, především proto, že svým výkonem a cenou, byla směřována do low-end sektoru trhu, a stala se tak slabší sestrou většího systému SGI Octane.

2.2 Hardware

Architektura

O2 je založeno na architektuře proprietarní širokopásmové Unifikované Paměťové Architektuře (Unified Memory Architecture, UMA), která slouží pro propojení jednotlivých částí systému (Procesor, grafický čip, operační paměť) viz následující obrázek:



Obrázek 1 – Schéma unifikované paměťové architektury[1]

Unifikovaná paměťová architektura stanice SGI O2 je založena okolo 4 banků a multiplexovanému SDRAM subsystému přinášejícímu propustnost až 2,1 GB/s. Každý ze subsystémů přistupuje k paměti skrz vysokorychlostní porty navržené tak, aby nedošlo k přehlcení informacemi na žádném s portů.

Co se týká hardwarové stavby SGI O2, tak celý systém je navíc navržen jako modulární, čímž se myslí, že jednotlivé komponenty stanice je možno jednoduše ze stanice vytáhnout, což velmi usnadňuje výměnu a údržbu jednotlivých komponent.

Pro rozšíření slouží jeden dostupný PCI slot, SCSI řadič, umožňující připojení až dvou disků (v případně použití procesoru R10000 nebo R12000 pouze 1), a především 1 slot pro přídavnou kartu pro připojení kamery nebo zvukových vstupů a výstupů.

Při označování systému SGI O2 se též velmi často užívá označení IP32, právě podle použité platformy, které je v našem případě při označování SGI O2 jedinečné.

Následující tabulka označuje stručný přehled součástí systému SGI O2 za níž následuje podrobný popis vybraných částí.

Tabulka 1

Přehled parametrů SGI O2	
Procesor	R5000 RM7000 R10000 R12000
Operační paměť	8 DIMM slotů 280pin SDRAM 64bitová sběrnice Až 1GB paměti
Pevný disk	Až dva (podle použitého chladiče procesoru) SCA UltraWide SCSI disk
Grafický čip	CRM grafický čipset
Síťový adaptér	10/100 Ethernet MACE MAC-110 Ethernet DP83840-1

Procesor

Následující informace jsou částečně převzaty a přeloženy z [2]

Systém O2 obsahuje procesor založený na architektuře MIPS - Microprocessor without Interlocked Pipeline Stages(mikroprocesor bez spřažených zřetězených stupňů) a může být založen na následujících dvou typech procesoru:

1. R5000/RM7000
    - Low-end procesor s frekvencí 180-350 MHz
2. R10000/R12000
    - High-end procesor s frekvencí 150-400 Mhz

Můj konkrétní model je založen na procesoru R10000 s frekvencí 175Mhz a 1024 kB L2 cache.

Více v části 2.4.2.

Operační paměť

Modul základní desky obsahuje 8 DIMM slotů pro až 1GB paměti se specifikací 280pin SDRAM DIMM připojených k 64 bitové sběrnici. Můj konkrétní model je vybaven 256 MB (8 slotů po 32 MB) paměti.

Pevný disk

Následující informace jsou částečně převzaty a přeloženy z [2]

O2 obsahuje UltraWide SCSI řadič (Adaptec 7880) na něhož můžou být připojeny dva SCA UltraWide SCSI disky. Jak již bylo zmíněno, při použití procesoru R10000 nebo R12000, však kvůli potřebě většího chladiče lze osadit jen 1 disk. Dále je na již zmíněný SCSI řadič připojena optická 4x CD-ROM mechanika, která je výměnná za jakýkoliv SCSI model mechanik, avšak kompatibilita je zaručena jen s mechanikami Toshiba.

V mém případě je obsažen disk IBM 4GB - FAST-10 WIDE SCSI.

Grafický adaptér

Následující informace jsou částečně převzaty a přeloženy z [2] *a* [3]

CRM grafický čipset vyvinutý speciálně pro SGI O2. Neobsahuje žádnou vlastní paměť, ale využívá prostředků hlavní operační paměti. A právě díky UMA architektuře a také 64bitové sběrnici to však nemá téměř žádné důsledky na výkon a navíc umožňuje načítání framebufferu přímo z hlavní paměti a tím je možné dosáhnout téměř neomezeného prostoru, omezeného jen velikostí paměti RAM, pro grafické operace s texturami, což vynikne především oproti dražšímu Octanu nebo Indigu2, které mají právě omezenou velikost paměti na textury řádově v jednotkách MB. Na druhou stranu při práci s 3D grafikou využívá pro výpočty také procesor a grafický čip má na starosti jen Z-buffer a podobně.

Pro práci s texturami a obrázky obsahuje SGI O2 ještě specializovaný procesor ICE (Image Compression Engine), který však dokážou využít jen přímo pro něj psané aplikace. Dále může být součástí stanice SGI O2 také VICE (Video Image Compression Engine) akcelerátor pro zpracování videa, který nám opět rozšiřuje možnosti jeho využití.

O2 též podporuje specifikaci OpenGL 1.1 s ARB rozšířením, avšak kvalit grafického adaptéru je možno využít zatím pouze pod OS Irix, jelikož pro ostatní systémy nejsou dostupné ovladače.

2.3 Operační systémy

Základním pro SGI O2 je unixový operační systém IRIX 6.3 a 6.5. Pro danou architekturu existuje také několik portů OpenBSD, NetBSD a linuxových OS, např. Debian a Gentoo.

2.4 Instrukční sada procesoru R10000

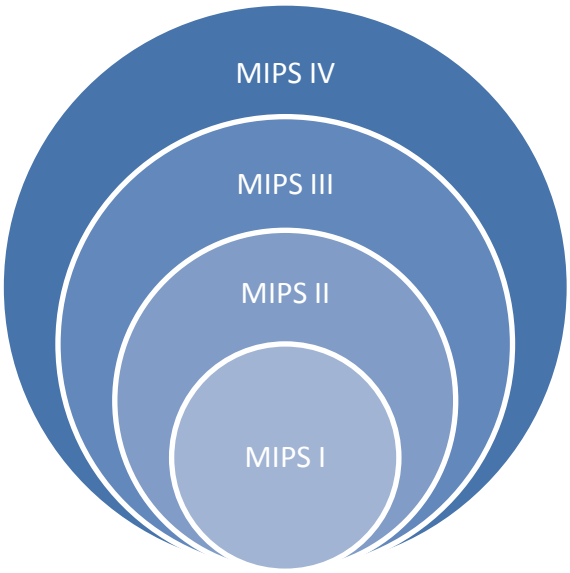
2.4.1 MIPS architektura

Následující informace jsou převzaty a přeloženy z [4]

Architektura MIPS definuje sadu instrukcí (instruction set architecture ISA) implementovanou v následujících procesorech:

MIPS I	R2000 a R3000
MIPS II	R6000
MIPS III	R4400
MIPS IV	R8000 a R10000

Originální MIPS I ISA byla třikrát rozšířena, je zobrazena na následujícím Obrázek 2 a každé její rozšíření je zpětně kompatibilní. To znamená, že procesor kompatibilní s MIPS IV architekturou dokáže spouštět programy pro předchozí architektury MIPS I-III.



Obrázek 2 - Vývoj instrukční sady MIPS [4]

2.4.2 Popis procesoru R10000

Následující informace jsou převzaty a přeloženy z [4]

R10000 je procesor jednočipový superscalarní RISC mikroprocesor, který je založen na MIPS RISC procesorové rodině a je potomkem procesorů R2000, R3000, R6000, R4400 a R8000.

R10000 používá MIPS ANDES(Architecture with Nonsequential Dynamic Execution Scheduling) architekturu.

Procesor má následující vlastnosti:

- Využívá 64 bitovou MIPS IV instrukční sadu
- Dokáže dekódovat 4 instrukce v každém pipeline cyklu, které se přiřazují do 3 instrukčních front
- Obsahuje 5 výpočetních pipeline, připojených na oddělené integer a floating point výpočetní jednotky
- Používá dynamické plánování instrukcí a podporuje vykonávání instrukcí mimo pořadí
- Používá „speculative branching“
- Využívá precizního modelu výjimek
- Obsahuje „non-blocking cache“
- Obsahuje oddělenou 32KB primární instrukční a datovou cache
- Má individuálně optimalizovanou L2 cache a „Systém interface port“
- Obsahuje interní řadič pro externí L2 cache
- Obsahuje interní řadič systémových rozhraní s podporou více procesorů

3 Testování paměti

Operační paměť je jedna z nejdražších a nejcitlivějších součástí citlivých na elektromagnetické výboje, a proto často například při manipulaci s ní dochází k jejímu poškození. Jelikož odhalení tohoto problému nemusí být vždy na první pohled zřejmé, bude se moje bakalářská práce zabývat právě tématem testování operační paměti.

Pro testování budu využívat algoritmy z aplikace Memtest86+, která je primárně zaměřena na zátěžové testy pamětí v systémech kompatibilní s x86 instrukcemi. Program testuje zápisy a čtení paměti v celém jejím rozsahu.

3.1 Memtest86+

Následující informace jsou převzaty a přeloženy z [5]

Program MemTest86+ zapisuje různé posloupnosti dat do paměti a testuje, zda po přečtení obsahují stejná data, která tam byla zapsána. Pokud nedojde ke shodě je pravděpodobně daná část paměti vadná. To může nastat z několika důvodů.

Nejčastější využití tohoto programu najdeme při testování stability a kompatibility pamětí s čipovou sadou nebo také při testování hranic možnosti chodu systému například při přetaktování, při snaze dosáhnout většího výkonu. V našem případě nás však zajímá především první zmíněné využití a to jako test funkčnosti stability a bezchybnosti paměťových modulů umístněných v systému SGI O2.

3.1.1 Historie Memtestu

Memtest86 byl vyvinut Chrisem Bradym a Memtest86+ Samuelem Demeulemeestrem.

Zavaděč je založen na Linuxu 1.2.1 a program je psán v jazyce C s využitím x86 asembleru. Použití zdrojových kódů umožňuje GNU General Public License. Nejnovější verze podporují dual i quad core procesory.

3.1.2 Spuštění

Start programu probíhá standardně z bootovatelného media například Diskety, CD-ROM, USB Flash disku nebo diskového oddílu, tedy bez nutnosti mít nainstalován operační systém. U SGI O2 nemáme na výběr příliš mnoho druhů médií, z kterých je možno bootovat. Primárně se jeví nejspolehlivějším mediem CD-ROM disk s mini-instalátorem viz v části 5.1.

3.1.3 Instalace memtestu

Překlad:

1. Kontrola Makefilu a upravení parametrů pro to co potřebujeme
2. Spuštění příkazu „make“

Tímto se vytvoří soubor „memtest.bin“, který je bootovatelný image. Tento soubor můžeme nakopírovat na disketu či použít zavaděč lilo a spouštět jej tak z diskového oddílu.

Vytvoření bootovatelné diskety:

1. Vložte prázdnou disketu s povoleným zápisem
2. Jako root zadejte příkaz „make install“

Nastavení diskového oddílu a bootování pomocí zavaděče lilo:

1. Nakopírujte image memtestu na pevný disk
2. Přidejte záznam do konfiguračního souboru zavaděče (obvykle /etc/lilo.conf), aby nabootoval memtest. Stačí, když bude specifikován image a název. Následující příkazy je třeba přidat do konfiguračního souboru.
   image = /memtest
   label = memtest
3. Jako root zadejte příkaz „lilo“
   V příkazovém řádku zavaděče lilo, nastavte bootování Memtestu86+.

Jestliže došlo k problémům při překladu, je možné vložit binární soubor „precomp.bin“.

K vytvoření bootovatelného disku s předpřipraveným imagem proveďte následující:

1. Vložte prázdnou disketu s povoleným zápisem
2. Zadejte příkaz „make instar-precomp“

3.1.4 Příkazy

Memtest86+ má několik příkazů, které umožňují kontrolu nad pamětí cache, výběrem testů, rozsahem adres nebo například výpisem chyb. Nápověda je zobrazena na spodní straně obrazovky.

Popis příkazů:

ESC Konec testů a softwarový restart počítače

c Vstup do konfiguračního menu

Možnosti menu jsou následující:

1. Cache mode – nastavení paměti cache
2. Test selection – výběr testů
3. Address Range – rozsah adres
4. Memory Sizing – velikost paměti
5. Error Summary – soupis chyb
6. Error Report Mode – výběr módu hlášení chyb
7. ECC mode – výběr pamětí s kontrolou chyb
8. Restart
9. Adv. Option – Další pokročilá nastavení

SPACE Nastavení blokování posunu okna (vypnutí posouvání při vypisování chyb)

ENTER Zruší blokování posunu okna (Zapnutí posouvání při vypisování chyb)

3.1.5 Chyby

Memtest má na výběr dvě možnosti hlášení chyb. Ve výchozím nastavení hlásí jednotlivé chyby, které nalezne, ale je možné přepnout do tzv. „BadRAM Patterns mode“, tedy režimu, při kterém je možné vytvoření Linuxového BadRAM vzoru paměti. Tato vlastnost umožňuje linuxu obejít vadné paměťové buňky a tím obsáhnou větší stability systému.

V individuálním režimu jsou následující informace zobrazeny na obrazovce, pokud dojde k nalezení chyby. Chybová zpráva je zobrazena jen v případě kdy došlo k nalezení špatné adresy nebo vadného bitu ve vzoru. Všechny hodnoty jsou zobrazeny hexadecimálně.

Tst:	Číslo testu
Failing Address:	Adresa, kde došlo k chybě
Good:	Očekávaný datový vzor
Bad:	Špatný datový vzor
Err-Bits:	Výjimka nebo offset očekávaných a špatných dat, ukazuje, na kterých pozicích byl nalezen chybný bit
Count:	číslo následujících chyb se stejnou adresou nebo špatným bitem

V režimu „BadRAM Patterns mode“ jsou jednotlivé řádky tisknuty ve formátu badram=F1,M1,F2,M2. Každý pár F/M, kde F representuje chybnou adresu a M je bitová maska pro danou adresu. Tento vzor nám říká, že byla nalezena chyba na adrese F a ve všech „1“ bitech v M. Tento vzor může obsahovat větší množství chyb, než se opravdu v paměti nachází, ale vždy bude obsahovat všechny, které v ní opravdu jsou. Tento vzor je navržen tak, aby dokázal jednoduše zachytit hardwarové chyby paměti pomocí poměrně strohé syntaxe.

BadRAM vzor je navržen pro spíše přidávání chyb něž jejich přehled. Počet párů je omezen na 5 z praktických důvodů. Ve výsledku je možno ručně vybrat ze vzorů, které jsou pro nás nejvýhodnější.

3.1.6 Délka testování paměti

Doba, po kterou se provádění testování paměti, závisí na několika faktorech. Především na rychlosti procesoru, rychlosti a také na velikosti paměti. Program Memtest86+ provádí testování paměti, dokud není zastaven a počítá počet správně provedených kompletních testů. Ve většině případů stačí provedení jednoho cyklu testů pro případné odhalení chyby, avšak pokud chceme získat jistotu, doporučuje se testovat paměti po delší dobu. Takto dokážeme především odhalit změny chování pamětí při dlouhodobé zátěži. Právě dlouhodobá zátěž zapříčiní nárůst teplot jednotlivých čipů, což může mít za následek nechtěnou změnu jejich chování. Dlouhodobé testování paměti se provádí řádově po několik hodin, případně dní.

3.1.7 Princip testování pamětí

Existuje mnoho způsobů jak testovat paměť. Jelikož mnoho testů používá různé datové vzory, avšak bez bližší znalosti testovaných pamětí a architektury, je jejich použití omezeno pouze na hledání závažných chyb paměti a nedokáže nalézt chyby způsobené přechodovým jevem. Základní test v BIOSu se tak například zbytečně snaží nalézt chyby způsobené tímto jevem.

Paměťové čipy obsahují obrovské pole pevně na sebe navazujících paměťových buněk pro každý bit dat. Obrovský význam má právě přechodný poruchový jev způsobený vzájemnou interakcí mezi paměťovými buňkami. Často tak dochází k tomu, že zapsání dat do jedné buňky způsobí změnu vedlejší buňky. Efektivní testy paměti se právě pokoušejí pracovat se znalostí tohoto jevu. Proto ideální strategie testování paměti by měla být následující:

1. Zápis nuly do první buňky
2. Zápis jedniček do všech ostatních buněk, jednou nebo vícekrát
3. Ověření zda první buňka stále obsahuje nulu

Je jasné, že tato strategie testování potřebuje ke svému správnému chodu znát, kolik paměťových buněk konkrétní čip obsahuje. Jak však zjistíme, vhledem k množství různých návrhů a výrobních postupů, se stává tato strategie prakticky nereálná. Nicméně existují algoritmy, které se dokáží této ideální strategii přiblížit.

3.1.8 Algoritmus

Memtest86+ používá dva algoritmy, které přinášejí téměř ideální strategii pro řešení testování paměti. První algoritmus „inverzní posun“ pracuje následujícím způsobem:

1. Zaplní paměť datovým vzorem
2. Začne test od nejnižší adresy
    a. Ověřuje, zda nedošlo ke změně vzoru
    b. Zapisuje doplněk k danému vzoru
    c. Inkrementuje adresu a opakuje 2a až 2c
3. Začne od nejvyšší adresy
    a. Ověřuje, zda nedošlo ke změně vzoru
    b. Zapisuje doplněk k danému vzoru
    c. Dekrementuje adresu a opakuje 3a až 3c

Tento algoritmus se dobře přibližuje ideálnímu testu paměti, avšak také má své limity. Většina dnešních paměťových čipů ukládá data o šířce od 4 do 16 bitů. U čipů, které mají větší šířku než 1 bit, je nemožné vybrat čtení nebo zápis pouze jednoho bitu. Tedy není možné garantovat, že budou testovány všechny okolní interakce. V tomto případě je nejlepší použít stejný vzor pro celou paměť, čím dokážeme eliminovat působení okolních buněk. Tedy použije se vzor samých jedniček nebo nul.

Také další jevy jako cachování (využívání rychlé vyrovnávací paměti) a bufferovní (dočasné ukládání dat do vyrovnávací paměti) a vykonávání instrukcí mimo pořadí překáží vyšší efektivitě algoritmu inverzního posunu.

Je možné sice vypnout využívání cache paměti, avšak vypnutí bufferu u nových vysoce výkonných čipů není možné.

Druhým algoritmem, používaným pro testování paměti, je Modulo-X, který není ovlivněn pamětí cache ani bufferem. Algoritmus funguje následujícím způsobem:

1. Pro počáteční rozsah od 0 do 20 se vykoná
    a. Zápis vzoru na každou 20. pozici v paměti
    b. Zápis doplňku vzoru na všechna ostatní místa v paměti
    … opakování 1b jednou nebo vícekrát …
    c. Kontrola každé 20. pozice v paměti

Tento algoritmus se nachází na stejné kvalitativní úrovni jako algoritmus inverzního posunu, avšak není limitován pamětí cache a bufferem. Protože je oddělen průchod zápisu (1a a 1b) a čtení (1c), zajišťuje použitelnost pro všechny paměti, jelikož buffery a paměť cache jsou pročištěny mezi každým dalším průchodem algoritmu. Výběr délky kroku je zcela libovolný, ale je třeba si uvědomit, že s narůstající délkou kroku sice dochází k větší přesnosti, však za cenu delšího běhu algoritmu. Výběr právě kroku 20 je kompromisem mezi přesností a rychlostí.

3.1.9 Popis jednotlivých testů

Memtest86+ rozděluje svůj proces testování do několika částí obsahujících specifické typy testů, které se snaží různými způsoby kontrolovat paměť. Tyto části obsahují kombinaci testovacích algoritmů, datových vzorů a cachování. Pořadí jednotlivých testů je nastaveno tak, aby dokázalo co možná v nejkratší době odhalit případné chyby. Dále následuje popis jednotlivých testů:

Test 0

Charakteristika: Test adresy, jeden průchod, není použita cache

Provádí test všech bitů paměti na všech paměťových modulech v systému za použití jednotného datového vzoru. Chyby nalezené v tomto testu nejsou použity do tzv. počítaných BadRAM vzorů.

Test 1

Charakteristika: Test adresy, vlastní adresa

Na každou adresu paměti se zapíše její adresa a poté se ověřuje její shoda. Teoreticky předchozí test odhaluje jakékoliv problémy při adresaci paměti. Tento test však dokáže odhalit problém při adresaci, na kterém předchozí metoda byla neúspěšná, tedy neodhalila chybu.

Test 2

Charakteristika: inverzní přesun, metoda jedna a nula

Tento test používá k přesunu inverzní algoritmus se vzory samých jedniček a nul. Také je aktivní cache, kterou využívají některé stupně testovacího algoritmu. Právě díky využití paměti cache dochází ke zkrácení času potřebného k vykonání tohoto testu a tím také rychleji dojde k odhalení „vážných“ chyb a také několika méně závažných chyb. Tato část je však jen rychlou kontrolou paměti.

Test 3

Charakteristika: inverzní přesun, 8bit

Tento test je stejný jako test 1, avšak používá 8 bitů široký datový vzor „chodících“ jedniček a nul. Umožňuje lepší detekci menších chyb v „širokých“ paměťových čipech. U tohoto testu je použito 20 datových vzorů.

Test 4

Charakteristika: inverzní přesun, náhodný vzor

Také tento test používá stejný algoritmus jako test 1, datový vzor obsahuje náhodná čísla a jejich doplňky. Tento test je obzvláště efektivní při hledání obtížně nalezitelných citlivých chyb. Celkem tento test používá 60 různých vzorů. Použitá náhodná čísla se mění každým úspěšných průchodem, čímž narůstá efektivita tohoto algoritmu.

Test 5

Charakteristika: Přesun bloků, 64 posunů

Tento test zatěžuje paměť blokovým přesunem dat (instrukce movsl) a je založena na testu burnBX Roberta Redelmeiera. Paměť je inicializována posuvným vzorem, který je převrácený každých 8 bajtů. Pote se 4MB blok paměti přesunuje dokola pomocí movsl instrukce. Poté co dojde k úplnému přesunu datového vzoru, ověří se shodnost dat. Jelikož se data ověřují až po dokončení přesunu dat, není možné přesně lokalizovat chybu. Adresa jen určuje, kde byl nalezen špatný vzor. Nežli dojde k přesunu na 8MB segment paměti, chybná adresa bude vždy v menší než 8MB vzdálenosti od nahlášené adresy, kde byl nalezen chybný vzor. Chyby z tohoto testu nejsou použity do počítaných BadRAM vzorů

Test 6

Charakteristika: inverzní přesun, 32 bitový vzor

Tato variace algoritmu inverzního přesunu mění ve vzoru jeden levý bit každé následující adresy. Počáteční bit pozice je posunut doleva po každém průchodu, a abychom použili možné datové vzory je potřeba alespoň 32 průchodů. Tento test je efektivní při hledání citlivých datových chyb, avšak jeho běh je poměrné dlouhý.

Test 7

Charakteristika: sekvence náhodných čísel

Tento test zapisuje do paměti sérii náhodných čísel. Pomocí resetování rozložení pro náhodná čísla, může být stejná sekvence použita také pro vytvoření kontrolního vzoru. Dojde k ověření počátečního vzoru a pak k jeho doplnění a poté je testován ještě jednou. Avšak na rozdíl od metody inverzního posunu, může zapisovat a kontrolovat pouze v dopředném směru.

Test 8

Charakteristika: modulo 20, 1&0

Použitím Modulo-X algoritmu můžou být odhaleny chyby, které nedokázal odhalit algoritmus inverzního posunu, kvůli interferencím algoritmu v paměti cache a bufferu. Samé jedničky a nuly jsou použity jako datové vzory.

Test 9

Charakteristika: Test slábnutí bitů, 90 minut, 2 vzory

Test slábnutí bitů (Bit fade test) inicializuje celou paměť vzorem a poté 90 minut čeká. Pak testuje, zda někde nedošlo ke změně bitů. Opět se zde používají vzory s jedničkami a nulami. K dokončení testu jsou zapotřebí 3 hodiny času. Test slábnutí bitů není běžně součástí normálního testu a musí se spouštět ručně pomocí konfiguračního menu.

4 Portace Memtestu na architekturu SGI/MIPS

Jak již bylo uvedeno v části „3.1.1“ program Memtest86+ je psán v jazyce C a využívá asembleru architektury x86. Proto je nutné, pokud bychom chtěli psát algoritmy v asembleru MIPS upravit části kódu psané v asembleru x86. Jde především o algoritmy jednotlivých testů a inicializace paměti. Rozhodl jsem se popsat především základní rozdíly architektur a popis adresace asembleru pro procesory MIPS.

4.1 Rozdíl architektur

Architektura x86 obsahuje CISCovou instrukční sadu oproti architektuře MIPS, která je RISCová. Procesor R10000, který je použit v systému SGI O2 a je dokonce 64 bitový a tak obsahuje odlišné uspořádání a velikosti registrů. Avšak je potřeba si všimnout, že právě procesor R10000 obsahuje mnoho vlastností, které se ustálily v procesorech architektury x86 až v poměrně nedávné době, jako například zpracování více instrukcí zároveň nebo vykonávání instrukcí mimo pořadí. Proto je dobré se zaměřit také na problémy, popsané v části 3.1.8 věnované algoritmu hledání paměti a upozorňující právě na problémy moderních architektur. Také je třeba se zaměřit na úpravu algoritmů, aby odpovídaly dané architektuře a tím se staly efektivními pro testování.

Největší rozdíly se však budou vyskytovat především v adresaci paměti, která je nejvíce závislá na volbě systému.

4.2 Přehled architektury – MIPS asembler

Následující informace jsou převzaty a přeloženy z [6]

4.2.1 Datové typy a literály

Datové typy:

- Všechny instrukce jsou 32 bitové
- Byte (8 bitů), halfword(2 bajty), word(4 bajty)
- Char má velikost 1 bajt
- Integer má velikost 1 word tedy 4 bajty

Literály:

- Číselné hodnoty se zadávají jako 4
- Char se zadává jako 'b'
- String jako “A string”

4.2.2 Registry

32 hlavních registrů

Registry jsou v asembleru označeny přidáním znaku $

Dva druhy adresace:

- Použití čísla registru - $0 až $31
- Použití ekvivalentního jména - $t1, $sp

Speciální registry Lo a Hi

- Slouží pro uložení výsledku násobení a dělení
- Nejsou přímo adresovatelné, ale přístupné pomocí instrukcí mfhi („move from Hi“) a mflo („move from Lo“)

Zásobník se adresuje od nejvyšší adresy po nejmenší

Tabulka 2 - Přehled registrů

Číslo registru	Alternativní název	Popis
0	Zero(nula)	Hodnota 0
1	$at	(dočasná hodnota) rezervováno asemblerem
2-3	$v0 - $v1	(hodnota) z výsledku výrazu nebo výsledek funkce
4-7	$a0 - $a3	(argument) první 4 parametry pro podprogram Hodnoty nejsou uchovávány při volání procedur.
8-15	$t0 - $t7	(dočasné hodnoty) Volající program je může použít pro ukládání hodnot. Podprogramy můžou použít ukládání w/out. Hodnoty nejsou uchovávány při voláním procedur.
16-23	$s0 - $s7	(uložené hodnoty) Volající ukládá hodnotu do tohoto registru. Podprogram používá jednu z těchto hodnot a musí ji uchovat a obnovit před svým koncem. Hodnoty jsou uchovávány při volání procedur.
24-25	$t8 - $t9	(dočasné hodnoty) Volající program je může použít pro ukládání hodnot. Zde jsou přídavné hodnoty pro $t0 - $t7. Hodnoty nejsou uchovávány při volání procedur.
26-27	$k0 - $k1	Rezervováno pro přerušení
28	$gp	Globální ukazatel Ukazuje do středu 64K bloků paměti na statický datový segment.
29	$sp	Ukazatel na zásobník Ukazuje na poslední místo v zásobníku
30	$s8/$fp	Ukazatel na Uloženou hodnotu/rámec Chráněný při volání podprogramů
31	$ra	Návratová adresa

4.2.3 Programová struktura

- Jen čistý text s deklarací dat, programovým kódem
- Část deklarací dat je následována programovým kódem

Deklarace dat

- Je umístěna v části programu obsahující příkaz v asembleru .data
- Obsahuje deklaraci proměnných používaných v programu, které jsou uloženy v hlavní paměti

Kód

- Je umístěn v části začínající .text
- Obsahuje programový kód (instrukce)
- Počátek kódu začíná od návěstí main:
- Konec části main může použít ukončovací systémové volání

Komentáře

- Všechno za znakem # je komentář

Šablona programu v MIPS asembleru:

```
# Zakladní struktura programu psaného v MIPS asembleru
     .data                 # deklarace proměných následuje za tímto
                           # řádkem
                           # ...
     .text                 # za tímto následují intrukce
                           # ...
main:                      # začátek kódu, první vykovávaná instrukce
                           # ...
# konec programu na konci by měl být prázdný řádek
```

4.2.4 Deklarace dat

Formát pro deklaraci dat:

```
jméno:      typ           hodnota
```

- Vytvoří prostor pro proměnou specifického typu s daným názvem
- Hodnota je obvykle počáteční hodnota; pro proměnou typu .space se vyhradí prostor dané velikosti

Poznámka: jméno musí být vždy ukončeno (:)

Příklad:

```
var1:       .word         3         # vytvoří proměnou integer a
                                    # zinicializuje ji na hodnotu 3
array1:     .byte         'a','b'   # vytvoří dva elementy pole char
                                    # inicializovaných na a a b
array2:     .space        40        # alokuje 40 sekvenčních bajtů,
                                    # které jsou neinicializované, ale
                                    # můžou být použity jakou pole 40
                                    # znaků nebo 10 integerů.
```

4.2.5 Load / Store Instructions

- Do paměti RAM mají přístup pouze instrukce Load a Store
- Ostatní instrukce používají registry

Načtení (load):

```
    lw      registr_cíl, RAM_zdroj
```

- kopíruje word(4 bajty) ze zdroje paměti RAM do cílového registru.

```
    lb      registr_cíl, RAM_zdroj
```

- kopíruje bajt ze zdroje v paměti RAM do cíle v registru (řazení od nejvyššího bajtu)

Uložení (store):

```
    sw      registr_zdroj, RAM_cíl
```

- uloží word z registru do paměti

```
    sb      registr_zdroj, RAM_cíl
```

- uloží bajt(řazení od nejnižšího) z registru do paměti RAM

Načtení hodnoty(load immidiate):

```
    li      registr_cíl, hodnota
```

- Načte do cílového registru danou hodnotu

příklad:

```
    .data
var1:       .word         23        # deklarace proměnné var1
                                    # inicializování hodnotou 23
    .text
__start:
    lw      $t0, var1     # načtení obsahu paměti RAM do registru $t0:
                          # $t0 = var1
    li      $t1, 5        # $t1 = 5   ("load immediate")
    sw      $t1, var1     # uložení obsahu registru $t1 do paměti RAM:
                          # var1 = $t1
    done
```

4.2.6 Nepřímé a bázové adresování

- Používájí ho jen load a store intrukce

Načtení adresy:

```
    la      $t0, var1
```

- Zkopíruje adresu proměnné var1 do registru $t0

Nepřímá adresace:

```
    lw      $t2, ($t0)
```

- Načte word z paměti RAM na adrese obsažené v registru $t0 do registru $t2

```
    sw      $t2, ($t0)
```

- Uloží word z registru $t2 do paměti RAM na adresu obsaženou v registru $t0

Bázové nebo indexované adresování:

```
    lw      $t2, 4($t0)
```

- Načte word z paměti RAM z adresy ($t0+4) do registru $t2
- „4“ definuje offset registru $t0

```
    sw      $t2, -12($t0)
```

- Uloží word z registru $t2 do paměti RAM na adresu ($t0 - 12)
- Offset může být i negativní

Poznámka: základní adresování se většinou používá pro adresaci následujícího:

- Pole, zpřístupnění položek pomocí offsetu z bázové adresy
- Zásobník, jednouchý přístup položek pomocí offsetu z ukazatele na zásobník (Stack pointer) nebo ukazatele na rámec (frame pointer)

Příklad:

```
    .data
array1:     .space      12      # deklaruje 13 bajtů prostoru
                                # pro 3 integery
    .text
__start:
    la      $t0, array1     # načte bázovou adresu pole do registru $t0
    li      $t1, 5          # $t1 = 5   ("load immediate")
    sw      $t1, ($t0)      # první položka pole nastavena na 5;
                            # nepřímá adresace
    li      $t1, 13         # $t1 = 13
    sw      $t1, 4($t0)     # druhá položka pole nastavena na 13
    li      $t1, -7         # $t1 = -7
    sw      $t1, 8($t0)     # třetí položka pole nastavena na -7
    done
```

4.2.7 Aritmetické instrukce

- Nejčastěji používá 3 operandy
- Všechny operandy jsou registry, neadresuje se RAM ani není možně použít nepřímou adresaci
- Velikost operandu je word (4 bajty)

Sčítání a odečítání:

```
add     $t0,$t1,$t2     # $t0 = $t1 + $t2;   součet znaménkový (2's
                        # complement) integerů
sub     $t2,$t3,$t4     # $t2 = $t3 - $t4
addi    $t2,$t3, 5      # $t2 = $t3 + 5;   "přímý součet" bez
                        # mezisoučtů
addu    $t1,$t6,$t7     # $t1 = $t6 + $t7;   součet bezznaménkových
                        # integerů
subu    $t1,$t6,$t7     # $t1 = $t6 - $t7;   rozdíl bezznaménkových
                        # integerů
```

Násobení a dělení:

```
mult    $t3,$t4         # násobení 32-bit hodnoty v $t3 a $t4, a
                        # uložení do 64-bit
                        # výsledku speciálním registru Lo a Hi:
                        # (Hi,Lo) = $t3 * $t4
div     $t5,$t6         # Lo = $t5 / $t6   (dělení integerů)
                        # Hi = $t5 mod $t6   (zbytek)
mfhi    $t0             # přesun hodnoty ze speciálního registru Hi
                        # do $t0:   $t0 = Hi
mflo    $t1             # přesun hodnoty ze speciálního registru Lo
                        #  do $t1:   $t1 = Lo
                        # používané na zjištění výsledku
                        # nebo hodnoty
move    $t2,$t3         # $t2 = $t3
```

4.2.8 Řízení struktury programu

Větvení:

- Porovnání pro podmíněné větvení se prování pomocí následujících instrukcí:

```
b       cíl             # nepodmíněné větvení na návěstí „cíl"
beq     $t0,$t1, cíl    # větvení na „cíl", jestliže $t0 = $t1
blt     $t0,$t1, cíl    # větvení na „cíl", jestliže $t0 < $t1
ble     $t0,$t1, cíl    # větvení na „cíl", jestliže $t0 <= $t1
bgt     $t0,$t1, cíl    # větvení na „cíl", jestliže $t0 > $t1
bge     $t0,$t1, cíl    # větvení na „cíl", jestliže $t0 >= $t1
bne     $t0,$t1, cíl    # větvení na „cíl", jestliže $t0 <> $t1
```

Skoky:

```
j       cíl             # nepodmíněný skok na návěstí „cíl“
jr      $t3             # skok na adresu obsaženou v $t3 ("registr
                        # skoku")
```

Volání podprogramu: "jump and link" instrukce

```
jal     návěstí_podprogramu     # "jump and link"
```

- Kopíruje programový čítač (návratová adresa) do registru $ra (registr návratových adres)
- Skáče na příkaz podprogramu na návěstí „návěstí_podprogramu“

Návrat z podprogramu: "jump register" instrukce

```
jr      $ra             # "jump register"
```

- skok na návratovou adresu z registru $ra (uložené pomocí jal instrukce)

Poznámka: Návratová adresa je uložena v registru $ra, jestliže podprogram volá další podprogram nebo dojde k rekurzivnímu volání, návratová adresa se zkopíruje z registru $ra na zásobník, aby se předešlo její ztrátě. Jal instrukce vždy uloží návratovou adresu do registru, a proto tak přepíše předchozí hodnotu.

5 Příprava vývojového prostředí

Jako linuxový operační systém jsem zvolil distribuci Debian pro systémy s architekturou MIPS. Zde je potřeba zmínit, že pro SGI O2 je třeba port IP32 pro procesor R5000.

5.1 Spuštěni instalátoru

Stanice SGI O2 disponuje na základní desce vestavěným operačním systémem UNIXového typu, který plní funkci konfigurace počítače (PROM). Po spuštění stanice se objeví okno „Starting up the system“ s tlačítkem „Stop for Maintenance“, po jehož stisknutí (stejného efektu lze docílit stiskem klávesy ESC), se zobrazí „obrazovka údržby stanice“-“System Maintenance Menu“.

Položky menu (lze volit číslicemi 1-6):

1. Start systému
2. Spuštění instalace systému
   - Vzdálená páska
   - Vzdálený adresář
   - Lokální jednotka CD-ROM
   - Lokální páska
3. Spuštění diagnostiky (vyžaduje nainstalovaný systém IRIX nebo CD-ROM se systémem IRIX)
4. Obnova systému
5. Spuštění konzole
6. Nastavení rozložení klávesnice

Jelikož linuxové distribuce podporující SGI O2, nejsou schopny úspěšně nabootovat systém z jednotky CD-ROM, výjimku tvoří Mini-Instalátor, bude většina následujících příkazů spuštěna přes konfigurační konzoli (volba č. 5, klávesa F5).

5.1.1 Instalační medium

Linux Debian je možno instalovat několika způsoby. Nejjednodušším způsobem je instalace z disku CD-ROM, jak již jsem zmínil pro SGI O2 to prozatím není možné. Je třeba tedy zvolit odlišný způsob instalace. Velmi vhodně se jeví použití Mini-instalátoru, který emuluje zdroj instalace jako instalátor systému IRIX. Tento instalátor lze například spustit ze sítě a poté již pokračovat v instalaci z jednotky CD-ROM.

5.1.2 Instalace přes síť

Pro následující část byly použity informace ze zdrojů [7], [8] a [9]

Jestliže jsme připojeni v síti, je možno ji využít jako zdroj instalace. Před spuštění instalace je vhodné, odstranit položku „netaddr“, příkazem:

```
unsetenv netaddr
```

Tento krok odstraní IP adresu stanice a přinutí systém k získání nové adresy z DHCP serveru.

Spuštění instalace ze sítě provedeme v BIOSu příkazem:

```
bootp():
```

Ten se pokusí kontaktovat DHCP server s podporou také BOOTP protokolu. Pro úspěšné přidělení adresy a komunikaci přes protokol „bootp“ je nutné zajistit, aby buď server ležel ve stejné podsíti, nebo pokud se nachází v jiné podsíti, bylo zajištěno přeposílání paketů z této sítě.

Nastaveni DHCP serveru pro protokol BOOTP

Pro úspěšné navázaní spojení se serverem je třeba správně nakonfigurovat DHCP server. Instalační proces využívá TFTP(Trivial File Transfer Protocol) a podstatné je také správné nastavení TFTP serveru.

Nastavení DHCP serveru provedeme v konfiguračním souboru (obvykle „/etc/dhcp3/dhcpd.conf“) například následovně:

```
host o2 {
     option tftp-server-name "10.0.0.5";
     next-server 10.0.0.5;
     filename "/tftpboot/o2-boot.img";
     hardware ethernet 08:00:69:05:ff:cb;
     fixed-address 10.0.0.90;
}
```

Po upravení konfiguračního souboru DHCP serveru je třeba provést jeho restart příkazem:

```
/etc/init.d/dhcpd3-server restart
```

TFTP server

Důležité je také povolit TFTP server. To můžeme provést 2 způsoby podle používaného programu. Pokud používáme **inetd**, provedeme zapsáním řádky do souboru „/etc/inetd.conf“

```
tftp dgram udp wait nobody /usr/sbin/tcpd in.tftpd /tftpboot
```

Druhou možností je zvolit alternativu v podobě **xinetd,** který jsem využíval, a upravit konfigurační soubor /etc/xinetd.d/tftp:

```
service tftp
{
     socket_type = dgram
     protocol = udp
     wait = yes
     user = root
     only_from = 10.0.0.90
     server = /usr/sbin/in.tftpd
     server_args = -s /tftpboot
     disable = no
     per_source = 11
     cps = 100 2
}
```

Kde jsem například ještě definoval, aby TFTP server odpovídal jen mému klientovi z IP adresy 10.0.0.90

Kopírování

K úspěšnému nakopírování souborů je třeba provést následující dva kroky:

Pokud hostitelský počítač obsahuje GNU/Linux s jádrem 2.4.X je třeba na něm vypnout „Path MTU discovery“:

```
# echo 1 > /proc/sys/net/ipv4/ip_no_pmtu_disc
```

Pokud tento krok neprovedete PROM, na SGI nebude schopen stáhnout jádro pro instalaci. Dalším problémem, se kterým se lze setkat, je nastavení správného rozsahu portů. Je třeba, aby server odesílal pakety na portu menším než 32767, jinak se přenos zastaví po vyslání prvního paketu.

Tento problém lze odstranit, změnou nastavení rozsahu portů, příkazem:

```
# echo "2048 32767" > /proc/sys/net/ipv4/ip_local_port_range
```

5.1.3 Zavedení systému

Zavaděčem systému Linux na architektuře MIPS je zavaděč „arcboot“. Tento zavaděč podporuje pouze souborové systémy EXT2 a EXT3. Jelikož původní souborový systém byl XFS, tak bylo nutné provést jeho přeformátování na EXT2.

Přeformátování proběhlo na stroji architektury i386, kde byl za použití řadiče SCSI připojen disk ze stanice SGI O2. Po zazálohování dat byl disk přeformátován na systém EXT2. Po opětovném vložení disku do stanice byl zjištěn problém se symbolickými odkazy na disku, a proto musely být zrušeny.

5.2 Vývojové prostředí na SGI O2

Po zavedení a instalaci systému linuxové distribuce Debian, jsem se rozhodl použít tuto jako vývojové prostředí pro překlad a testování programů na architektuře MIPS – stanici SGI O2.

Než jsem však začal psát programy, rozhodl jsem se vyzkoušet jaké možnosti mně právě stanice SGI O2 dává a vyzkoušet nainstalovat X-server.

Grafické prostředí X-server jsem nainstalovat pomocí jednotlivých balíčků **xorg**. Po vyzkoušení funkcionality X-serveru, jsem pokračoval v instalaci grafického rozhraní KDE, kteréžto se mně stalo mým pracovním prostředím, a které je schopno na systému SGI O2 běžet v rozlišení 800x600 při 15 bitové barevné hloubce. Tímto jsem si připravil prostředí pro testování aplikací na SGI O2.

5.3 Přechod ze SGI 02 na křížový kompilátor[1]

V předchozích částech jsem se zabýval postupem pro instalaci Linuxu na SGI O2, jakožto prostředí pro vývoj aplikací. Bohužel tento postup obnáší celou řadu obtíží, ať již volnou diskovou kapacitou, rychlostí překladu na stanici SGI O2, a především ne 100% stabilitu systému.

5.3.1 Chyby na SGI O2

Při práci na astanici SGI O2 pod OS Debian jsem narážel poměrně často na následující chybu, která se objevovala při spouštění aplikací:

```
CRIME CPU error at 0x01005fcc0 status 0x00000004
```

Dalším problémem, který čas od času nastal, byla částečně nefunkční myš, což se, v prostředí PROM, projevuje buď jejím úplným zamrznutím nebo velmi omezeným pohybem v rámci horní části obrazovky. Tento problém je však poměrně lehce odstranitelný vysunutím a opětovným zasunutím konektoru myši.

[1] Pojem křížový kompilátor převzat z http://cs.wikipedia.org/wiki/Křížová_kompilace

5.3.2 Křížový kompilátor

Kvůli předchozím problémům jsem se pro vývoj mého programu rozhodl pro nasazení virtuálních stanic, nezávislých na hostovském systému, jelikož primárně používám systém MS Windows Vista, a tím pádem možnosti využít výpočetní kapacitu, diskový prostor dnešních počítačů a další výhody, které vizualizace přináší. Jako vývojové prostředí jsem využil, již díky zkušenosti ze SGI O2, linuxovou distribuci Debian. Nyní zde uvedu postup pro zprovoznění křížového kompilátoru (cross-compiler), pokud možno nezávisle na zvolené distribuci.

Citace pojmu křížový kompilátor z [10]*:*

> *Křížová kompilace je kompilace, při které kompilátor generuje kód spustitelný na jiné platformě, než na které je kompilátor spuštěn. Nástroje pro křížovou kompilaci jsou nejčastěji používány pro generování spustitelných souborů pro Embedded systémy a pro různé platformy. Jsou používány pro kompilaci pro platformy, které nejsou schopny kompilace, jako jsou například jednočipové počítače, které nemají operační systém. Dochází tak k usnadnění vývoje aplikací, například pro mobilní telefony. Křížová kompilace se také používá pro kompilaci pro systémy, které mohou mít více platforem - pomocí virtuálních strojů Paravirtualization). Například kompilace pro jazyk JAVA.*

Pro překlad programů spustitelných na MIPS procesorech je nutné provést především instalaci binutils pro MIPS, konkrétně jsem zvolil universálnější postup překladu zdrojových kódů ve verzi binutils_2.18.1 (binutils_2.18.1~cvs20080103.orig.tar.gz[2]).

1. Je třeba rozbalit zdrojové kódy do námi vybraného adresáře, např. /data/install/binutils
2. Poté je třeba ve složce, kde se nachází hlavní Makefile spustit příkaz:

```
./configure –prefix=usr/local/ --target=mips-linux
```

3. Potom příkazem make se utility přeloží a make install nainstalují

Tímto dostaneme potřebné utility pro instalaci překladače. V dalším kroku tedy provedeme překlad a instalaci překladače (gcc-core-4.3.3.tar.bz2[3]).

1. Opět rozbalíme archiv do příslušné složky, např. /data/install/gcc
2. Vytvoříme složku mips-linux[4], ve které spustíme následující příkaz:

```
../configure –prefix=usr/local/ --target=mips-linux
```

Poznámka: pro konfiguraci a překlad gcc je třeba mít naistalováno GMP nebo MPFR

3. Poté příkazem make přeložíme Gcc a *make install* nainstalujeme.

Poznámka: Při překladu bude pravděpodobně nutné dohrát několik hlavičkových souborů knihoven. V případě problémů je dobré použít aplikace strace, která monitoruje systémové volání.

Po konfiguraci, překladu a instalaci gcc je již možno provádět překlad programů pro MIPS, jen je třeba upravit makefile soubory programů, aby využívaly křížového překladače, čehož docílíme přidáním předpony **mips-linux-** před požadovaný název programu.

Dále v mojí práci již budu předpokládat použití křížového kompilátoru.

---

[2] http://ftp.de.debian.org/debian/pool/main/b/binutils/binutils_2.18.1~cvs20080103.orig.tar.gz
[3] http://ftp.icm.edu.pl/packages/gcc/releases/gcc-4.3.3/gcc-core-4.3.3.tar.bz2
[4] Složku mips-linux zde vytváříme, abychom se vyhnuli případným problémům při překladu překladače. Další výhodou je také, že pokud experimentujeme s překladačem, můžeme takto jednoduše vše promazat a začít znovu.

6 Praktická část – vytvoření programu

Pro vytvoření programu pro test paměti a jeho správnou funkčnost je třeba dobře zvážit prostředí, ve kterém bude testování paměti probíhat. Moje práce se dělí do několika fází, ve kterých jsem otestovat právě závislost testů na prostředí, ve kterém jsou spuštěny, a pokusil jsem se nastínit ideální prostředí pro testy. Ideálním cílem by bylo spustit test v prázdné paměti, avšak tento problém je netriviální s několika důvodů a také k němu vede více méně komplikovaná cesta. Moje práce se bude snažit ukázat výsledky, kterým jsem při cestě k tomuto cíli dosáhl.

Ještě než začnu popisovat jednotlivé fáze mojí práce, tak bych rád objasnil jakým způsobem se spouští stanice SGI O2. Jednotlivé fáze postupu jsou vyobrazeny na Schéma 1. V zelené fázi, po stisku tlačítka Start, dojde ke startu stanice a interní kontrole a inicializaci hardwaru. Pokud tato fáze

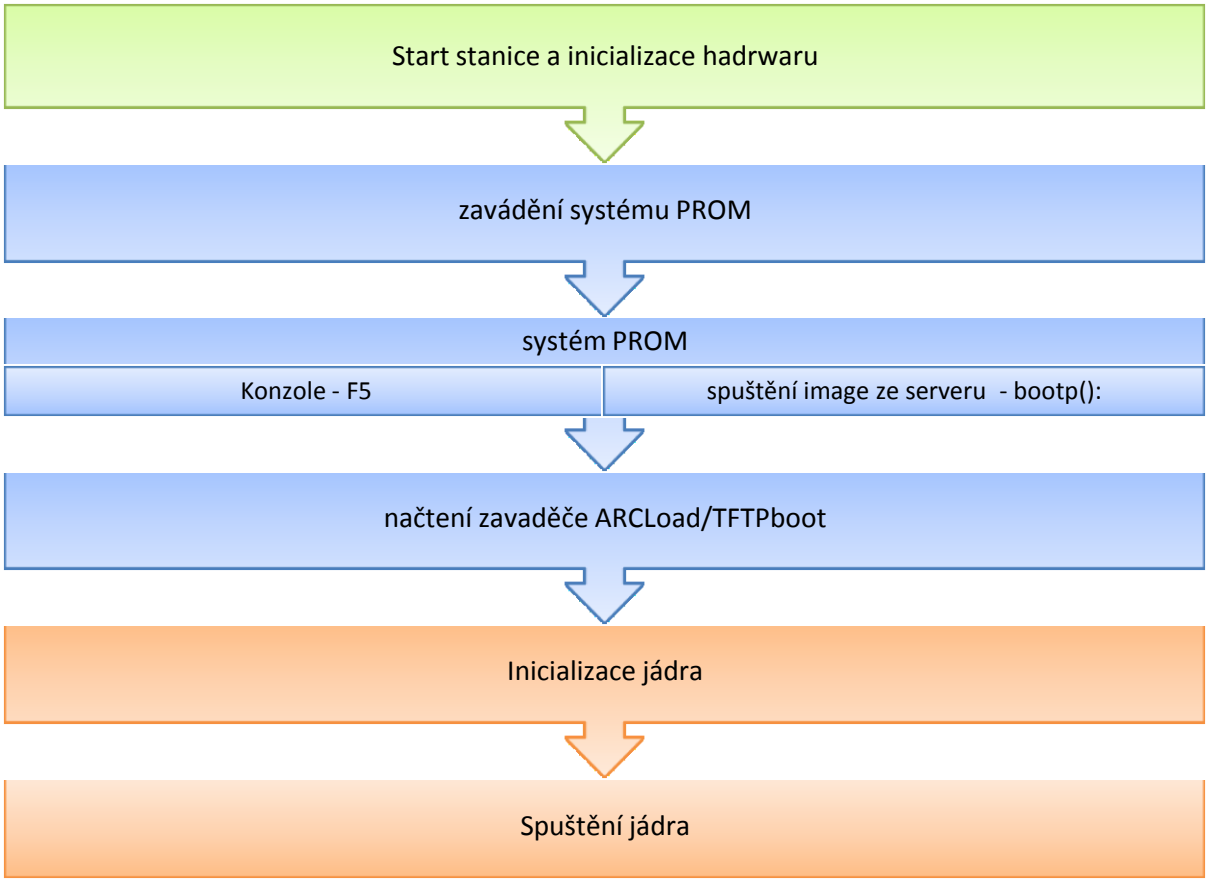


Schéma 1 - Start stanice SGI O2

dopadne dobře přejde se do modré fáze - zavádění systému PROM, který se pokusí spustit OS z pevného disku, nebo můžeme přejít do tzv. servisního menu, kde nás bude nejvíce zajímat konzole a spuštění bootování přes síť, viz kapitola 5.1. Po stáhnutí image souboru se zavaděčem a jádrem a spuštění zavaděče se přejde do červené fáze, kde již proběhne inicializace a spuštění jádra Linuxu.

6.1 Test paměti v prostředí PROM

V první fázi jsem se zaměřil na vytvoření programu v zavaděči, který je spuštěn z vestavěného unixového prostředí PROM. Této fázi předcházel postup z části 5, oproti níž jsem však místo image s instalátorem, zavedl můj program s testem paměti.

Program je psán v jazyce C a je třeba ho přeložit jako spustitelný program ve formátu ECOFF pro MIPS procesory. To je možno provést buď přímo na SGI O2 nebo můžeme použít křížový kompilátor, jehož instalace a konfigurace je popsána v části 5.3.2 Křížový kompilátor.

6.1.1 ARCLoad

Pro následující část bylo použito informací ze zdroje [11]

Jak sem již popisoval, v první části mojí práce jsem se zaměřil na úpravu zavaděče, konkrétně jsem zvolil ARCLoad, který je určen pro zavádění jádra Linuxu na stroje SGI/ARCS a jedná se prakticky o program spuštěný v prostředí PROM. Tato aplikace může využívat možnosti již spuštěného prostředí.

V současné době tento zavaděč podporuje následující systémy:

- **IP22**: SGI Indy, SGI Indigo2, SGI Challenge S
- **IP27**: SGI Origin200, SGI Origin2000, Onyx2
- **IP28**: SGI Indigo2 R10000
- **IP30**: SGI Octane, SGI Octane2
- **IP32**: SGI O2

ARCLoad je založený částečně na knihovnách od SGI usnadňujících práci s prostředky prostředí PROM. Osobně jsem použil zdrojové soubory upravené Stanislawem Skowronkem v roce 2005(arcload-0.5.tar.bz2[5]). Zavaděč jako takový obsahuje, jak načtení knihoven SGI, tak také samozřejmě prostředky pro zavedení jádra Linuxu a to zavaděč ARCGRUB, který dokáže spustit jádro z několika různých médií.

Pro moji práci však, v této první fázi, má význam především spuštění potřebných knihoven pro práci s pamětí a ovládání prostředí PROM.

6.1.2 Chování aplikace v prostředí PROM

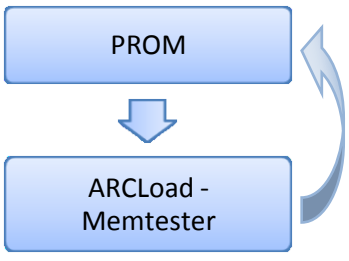
Jestliže pracujeme a spouštíme aplikace v prostředí PROM, můžeme využívat již mechanizmů spuštěného OS unixového typu, který se například dokáže vyrovnat s havárií naší aplikace, a pokud nebyla kritická, tak se dokáže vrátit do konzole nebo v případě například špatného zápisu do registrů procesoru vyvolá chybu a nabídne restart stanice.

Aplikace spuštěná v konzoly prostředí PROM můžou, pomocí funkcí obsažených ve zdrojovém souboru „**arc.c**“, využívat možností a měnit proměnné konzole a tím ovládání základní funkcionality stanice SGI O2.

6.1.3 Memtester a testování

Memtester je moje vlastní aplikace psaná v jazyce C, která je založena na zavaděči ARCLoad a určena především k demonstraci možností prostředí PROM a je také přípravou algoritmů pro testování paměti. Aplikaci tvoří základní menu, kde je možno nakonfigurovat chod testů, a také demonstruje některé možnosti prostředí PROM. Obsažené testy paměti jsou však omezeny právě tímto prostředím.

Hlavní problémem této části je především nemožnost přesně zjistit v jakém stavu se nachází paměť stanice a též jsme omezeni nemožností zasahovat do oblasti paměti vyhrazené systému PROM.



Obrázek 3 - Program Memtester se spouští jako aplikace v konzoly prostředí PROM, a tedy po jejím ukončení se opět vrátí zpět do konzole.

[5] ftp://ftp.linux-mips.org/pub/linux/mips/people/skylark/arcload-0.5.tar.bz2

Testovací stanice

Pro účely testování jsem použil 2 stanice SGI O2, ke kterým se v následující tabulce nachází základní popis rozdílů:

Tabulka 3 - Přehled použitých stanic

Název	LAMBDA	FERA
Platforma	SGI O2 – ip32	SGI O2 – ip32
CPU	R10000 – 194 MHz	R10000 – 174 MHz
RAM	256 MB	256 MB

U výsledků testů se tedy budu vždy odkazovat na konkrétní stanici, na které byly výsledky naměřeny.

Zjištění dostupné paměti

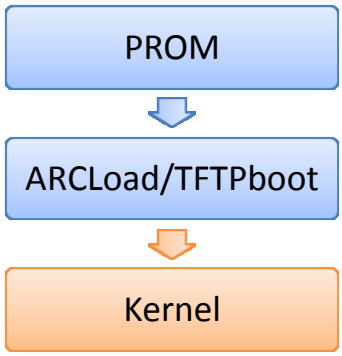
Ke zjištění dostupné paměti jsem použil algoritmus pro alokaci paměti z programu ARCLoad, který jsem se pokusil upravit, aby se pokoušel alokovat veškerou dostupnou paměť. Při pokusu alokovat paměť na stanici LAMBDA bylo zjištěno přibližně 235 MB volné paměti RAM pro ostatní aplikace spuštěné v prostředí PROM.

Konfigurace programu

Jak bylo zmíněno, program obsahuje možnost konfigurace, jak jednotlivý testů tak také délky jejich průběhu. Další zajímavostí je také experimentální možnost zvolit si velikost testované paměti, kde AGRESIVE mód umožňuje testovat paměť až okolo hranice maximální volné paměti, což může mít v jistých situacích vliv na paměť pro aplikace spuštěné v prostředí PROM, je tedy možno si přepsat paměť využívanou právě programem Memtester.

6.2 Test pamětí v jádře Linuxu

Druhá fáze mé práce se zaměřila na to, zda by bylo možno nějakým způsobem zlepšit testování paměti a odpoutat se právě od vestavěného prostředí PROM. Pro tuto fázi jsem předpokládal, že po zavedení jádra zavaděčem je v tuto chvíli paměť až na zavedené jádro jinak volná.



Obrázek 4 - Z prostředí PROM se spustí zavaděč jádra a poté dojde k inicializaci a spuštění linuxového jádra

6.2.1 Vytvoření zavaděče

Primárním úkolem druhé fáze je vlastní zavedení jádra linuxu. Přestože se tento úkol z počátku nezdál tak obtížný, nalezení správného zavaděče a následné vytvoření bootovatelného image se ukázal jako jeden ze zásadních problémů. Celý proces hledání zavaděče a způsobu jak vytvořit bootovatelný image je možno rozdělit od 3 částí:

1. Zavaděč ARCLoad a skript pro vytvoření boot CD
2. Debian-installer
3. Tip22

Zkusím zde popsat, kde jsem se dostal v každé z těch fází při hledání vhodného zavaděče a jak sem dospěl k tomu, že se nakonec jako nejpoužitelnější stala především až poslední fáze Tip22.

Zavaděč ARCLoad a skript pro vytvoření boot CD

Pro následující část bylo použito informací ze zdroje [12]

Původní myšlenka byla použít zavaděč z fáze jedna a použít jej jako základ image souboru. Původní má práce se tedy zaměřila na úpravu skriptu pro vytvoření bootovatelného CD pro SGI O2.

Tento skript zpočátku dokáže vytvořit prázdný image soubor příkazem:

```
dd if=/dev/zero of=$IMAGE bs=1M count=$IMAGE_SIZE
```

kde $IMAGE je název souboru a $IMAGE_SIZE velikost image.

V další fázi je třeba vytvořit oddíly v tomto image souboru a to za použití programu **parted**. Zde však platí pravidlo, že aby bylo možné s image bootovat, tak volhdr[6] musí být oddíl 9 a data oddíl 8.

Prvně je však třeba image pojmenovat příkazem:

```
parted -s $IMAGE mklabel dvh
```

kde opět jeho jméno tvoří $IMAGE

Poté vytvoříme 7 oddílů, aby bylo možno vytvořit datový oddíl a oddíl zavaděče. Pokud máme vytvořeny oddíly pro data (souborový systém ext2, ext3 parted neumí) a volhdr (je třeba vytvořit rozšířený oddíl) tak můžeme pomocné oddíly vymazat, volhdr(do 1MB) a datový oddíl (podle účelu použití pro samotné jádro stačí do 10MB) zvětšit na požadovanou velikost a naplnit volhdr zavaděčem příkazem:

```
dvhtool -d $IMAGE --unix-to-vh ext2load arcboot
```

dále by poté bylo třeba ještě nakopírovat potřebná data do datového oddílu.

Bohužel se však metoda 1. fáze potýkala s mnoha problémy, především se slinkováním jednotlivých částí dohromady a správným nastavením zavaděče, a po několika neúspěšných pokusech ladění, kdy se touto metodou nedařilo úspěšně zavést zavaděč a spustit jádro, jsem se rozhodl ji opustit a přejít do fáze 2.

Debian-installer

Pro následující část bylo použito částečně informací ze zdroje [13]

K fázi 2. jsem přistoupit po emailové korespondenci s Lukem Claesem z Debianu, u kterého jsem právě nalezl log (build_r5k-ip32_netboot-2.6.log[7]), který vznikl při kompilaci síťové instalace

[6] Volhdr neboli volume header je malý oddíl na začátku každého SGI disku. Obvykle zabírá okolo 2MB, do kterých by se měly vejít soubory potřebné pro start systému. Obsahuje potřebné informace o disku a také může obsahovat programy, které se mají spustit ještě před startem systému.[15]

[7] http://people.debian.org/~luk/mips/images/daily/build_r5k-ip32_netboot-2.6.log

distribuce Debian(netboot-boot.img[8]). Luk Claes mě odkázal na SVN repositář, konkrétně na Debian-installer.

Debian-installer je program vytvořený a podporující linuxovou distribuci Debian a je schopný vytvořit různé verze instalačních médií od CD, DVD a různých mini-instalací, také instalace síťové. Rovněž podporuje rozmanitou škálu architektur: ALPHA, AMD64, ARMEL, HPPA, IA64, I386, MIPS, MIPSEL, M68K, POWERPC, PPC64, SPARC a S390. Z mého pohledu je samozřejmě nejzajímavější architektura MIPS, kde debian-installer podporuje následující pro mě zajímavé sub-architektury: r4k-IP22(Indy), r5k-IP32(O2).

Jelikož však Debian-installer, je přímo závislý na linuxové distribuci Debian a také obsahuje poměrně velký balík funkcí, jsou jemu úměrné také nároky na knihovny a závislosti, bez kterých není možný běh tohoto programu.

Na architektuře MIPS, konkrétně r5k-IP32, využívá Debian-installer pro vytvoření síťové instalace zavaděče ARCBOOT a za zásadní součásti ohledně zavaděče může skript tftpboot.sh. A právě tento skript se stal zásadní pro přechod do následující fáze vytvoření bootovatelného síťového image. Zde se totiž zmiňovaný skript odkazuje na program pro vytvoření bootovatelného image souboru tip22.

Tip22

Pro následující část bylo částečně použito informací ze zdroje [14]

Další a již poslední fáze při hledání „jak vytvořit vlastní bootovatelný image“ se točí kolem programu tip22, což je Linuxový TFTPboot zavaděč.

Tip22 se používá k sestavování jádra a initrd (dočasný souborový systém v paměti RAM - ramdisk) do jednoho bootovatelného image souboru. Tento image také obsahuje malý zavaděč sloužící ke zkopírování jádra na jeho zaváděcí adresu a přichystání init ramdisku. Obvykle se používá právě pro bootování po síti před tftp nebo také z disku CDROM. Tento program je cílen především na sub-architektury IP22 a IP32 tedy SGI Indy a SGI O2.

Co se týče závislostí, potřebuje ke svému chodu především **binutils**, který obsahuje linker, a **file**, program pro určení typu souboru s pomocí tzv. "magických" čísel.

K přípravě programu tip22 nám poslouží archiv se zdrojovými kódy programu arcboot (např. arcboot_3.12.tar.gz[9]), který rozbalíme a upravíme následující soubory. Předpokládejme také, že jsme obsah archivu rozbalili do složky /data/zavadec/arcboot/.

K vytvoření bootovatelného image na sub-architekturu IP32(SGI O2) budeme používat skript tip32, který provede všechny přípravné operace a vytvoření bootovatelného image souboru. Tento skript se nachází v adresáři /data/zavadec/arcboot/tip22/. Je však potřeba jej upravit následujícím způsobem a dodat mu potřebné soubory pro jeho správný chod:

1. Je potřeba změnit proměnné ve skriptu tip32 podle následující tabulky:

Originál	Upravený
`SUBARCH=IP32`	`SUBARCH=IP32`
`LIBDIR=/usr/lib/tip22`	`LIBDIR=./`
…	…
`LD=/usr/bin/ld`	`LD=/usr/local/bin/mips-linux-ld`

[8] http://ftp.nl.debian.org/debian/dists/lenny/main/installer-mips/current/images/r5k-ip32/netboot-boot.img
[9] http://ftp.de.debian.org/debian/pool/main/a/arcboot/arcboot_0.3.12.tar.gz

2. Dále můžeme stáhnout balíček tip22_0.3.12_mips.deb, čímž si můžeme zjednodušit překlad programu arcboot, a rozbalíme ho příkazem:
   `dpkg -x tip22_0.3.12_mips.deb /data/zavadec/tip22_deb`
3. V adresáři tip22_deb/usr/lib/tip22 najdeme soubory, potřebné k vytvoření bootovatelkého image. Ještě než nakopírujeme potřebné soubory, pojďme se podívat jaké změny se nacházejí v konfiguračních skriptech(/data/zavadec/atcboot/tip22):

   a. Je potřeba změnit proměnné v souboru ld.kernel.script.in podle následující tabulky a přejmenovat ho na ld.kernel.script.IP32:

Originál	Upravený
`OUTPUT_FORMAT("@@OUTPUTFORMAT@@")`	`OUTPUT_FORMAT("elf32-tradbigmips")`

   b. Je potřeba změnit proměnné v souboru ld.ramdisk.script.in podle následující tabulky a přejmenovat ho na ld.ramdisk.script.IP32:

Originál	Upravený
`OUTPUT_FORMAT("@@OUTPUTFORMAT@@")`	`OUTPUT_FORMAT("elf32-tradbigmips")`

   c. Je potřeba změnit proměnné v souboru ld.script.in podle následující tabulky a přejmenovat ho na ld.script.IP32:

Originál	Upravený

`OUTPUT_FORMAT("@@OUTPUTFORMAT@@")`
`OUTPUT_ARCH(mips)`
`ENTRY(_start)`
`SECTIONS`
`{`
`/* XXX: place the loader after the kernel */`
`. = @@LOADADDR@@;`
	`OUTPUT_FORMAT("elf32-tradbigmips")`
	`OUTPUT_ARCH(mips)`
	`ENTRY(_start)`
	`SECTIONS`
	`{`
	`/* XXX: place the loader after the kernel */`
	`. = 0x81404000;`

4. Nyní již máme upraveny konfigurační soubory a ještě je třeba překopírovat soubory libarc.a a tftpload.IP32.o do adresáře se skriptem tip32.sh. Tím bychom měli všechny základní potřebné soubory pro úspěšné spuštění tohoto skriptu a můžeme se podívat na soubory, které se zadávají jako parametry.

5. Spuštění skriptu tip32.sh má následující syntaxi:

   `tip32 jádro ramdisk výstupní_soubor`

   a. Jádro – musí být ELF 32 nebo 64bit MSB spustitelný soubor pro architekturu MIPS, MIPS-IV

      Soubor: vmlinux

   Poznámka: Vytvořením vlastní kompilace jádra se zabývá další část 6.2.3

   b. RAM disk – pro naše účely téměř libovolný, např. textový soubor zabalený programem gzip, např. ramdisku.gz, jelikož jeho velikost i obsah je pro nás v tento okamžik nedůležitý, protože k jeho využití dochází až později při spouštění jádra.

   c. Výstupním souborem bude v našem případě např. netboot_my.img

6. Nyní bychom již měli mít všechny potřebné soubory pro úspěšné vytvoření bootovatelného image souboru s vlastním jádrem a můžeme spustit skript tip32:

   `./tip32 vmlinux ramdisku.gz netboot_my.img`

   Tento příkaz nám vytvoří soubor netboot_my.img, který již můžeme umístnit do námi zvolené složky pro bootování např. /tftpboot/

Tímto se nám podařilo úspěšně vytvořit bootovatelný image souboru s vlastním jádrem Linuxu. V Dalších kapitolách se budu zabývat právě tím, jaké jádro zvolit, jak ho upravit a co nejvíce zmenšit.

6.2.2 Hledání jádra

Jedním z velmi podstatných problémů při použití vlastního jádra je zvolení právě takové verze, která je buď již upravená nebo je možno ji upravit, tak aby byla spustitelná na architektuře MIPS. Prvně jsem se snažil využít nejnovějšího dostupného jádra Linuxu. Což však nevedlo k valnému úspěchu a tak jsem zvolil trochu starší verzi 2.6.28.4[10] upravenou právě pro architekturu MIPS, kterou již bylo možno na mojí stanici spustit. Soubor linux-2.6.28.4.tar.gz rozbalíme např. do adresáře /data/jadro/linux-2.6.28.4/ .

6.2.3 Úprava jádra

Pokud již máme nalezeno to správné jádro Linuxu, můžeme přejít do další fáze. Ještě jednou je dobré si připomenout otázku „proč vlastně potřebujeme jádro Linuxu?“. V předchozích částech jsme si řekli, jak vypadá testování paměti v prostředí PROM a proč nás právě toto prostředí omezuje. A právě omezení velikosti dostupné paměti, především již spuštěným prostředím PROM, bylo hlavní příčinou, proč se zabýváme hledáním řešení přes jádro Linuxu, kde právě můžeme získat, až na zavedené jádro, které je o poznání menší než celé předchozí prostředí, zbývající dostupnou paměť naší stanice.

Nyní se tedy omezujícím prvkem stává velikost jádra, která je však již námi ovlivnitelná. A právě v této kapitole se budu zabývat procesem zeštíhlování jádra Linuxu.

Ještě než se začneme zabývat konfigurací jádra, je třeba zkontrolovat a případně upravit Makefile soubor.

1. Proměnou ARCH:

```
ARCH                    ?= mips
```

2. Pokud používáme křížový kompilátor, musíme změnit proměnou CROSS_COMPILE následujícím způsobem:

```
CROSS_COMPILE  ?= mips-linux-
```

Můžeme také nastavit tuto hodnotu jejím zapsáním před příkaz make:

```
CROSS_COMPILE=mips-linux- make
```

Konfigurace jádra

Jádro Linuxu obsahuje mnoho rozličných částí, ze kterých právě ty nejdůležitější, z pohledu ořezávání a úpravy jádra, bych zde rád popsal.

Začnu od částí, která se nachází v adresáři arch(/data/jadro/linux-2.6.28.4/arch/), která právě obsahuje součásti jádra závislé na jednotlivých architekturách. Mnou zvolené jádro Linuxu obsahuje podporu pro velké množství rozličných architektur[11]. Jak jsem již zmiňoval na začátku, budeme se zabývat sub-architekturou IP32, pro kterou můžeme jádro zkonfigurovat pomocí příkazu, který spustíme v kořenovém adresáři zdrojových souborů jádra (/data/jadro/linux-2.6.28.4/):

```
make ip32_defconfig
```

[10] http://www.linux-mips.org/pub/linux/mips/kernel/v2.6/linux-2.6.28.4.tar.gz

[11] DEC Alpha, ARM, AVR32, Blackfin, ETRAX CRIS, FRV, H8/300, IA-64, MIPS, mn10300, M32R, m68k, m68knom mu, PA-RISK, PowerPC, SuperH, SPARC, SPARC64, S390, Xtensa, X86

Tímto se nám podařilo zkonfigurovat jádro pro grafickou stanici SGI O2, a můžeme ho přeložit příkazem **make** a použít (konkrétně soubor **vmlinux**) do námi připraveného image souboru z kapitoly 6.2.1 část. Tip22

Dále je již možno provádět jednotlivé odlehčování jádra Linuxu podle našich potřeb. K tomuto účelu jsem použil příkaz:

```
make menuconfig
```

Který umožní použít uživatelsky přívětivější prostředí pro výběr jednotlivých položek jádra. Ke spuštění tohoto příkazu je však nutné mít nainstalovánu knihovnu ncurses-devel (např. libncurses5-dev[12]).

V toto prostředí umožňuje přehledně nastavovat jak jednotlivé části jádra tak také přináší možnost vytváření i jiných konfiguračních souboru než „.config“. Je tedy možno jednotlivé konfigurace podle libosti ukládat a znovu načítat.

Nyní se podívejme jaké jednotlivé části (oblasti), které můžeme v konfiguraci jádra nastavit. Uvedu zde nastavení všech položek, které jsem ponechal zapnuty (pokud je jinak, je uvedeno). Neznamená to však, že se zde nacházejí jen nejnutnější části jádra, ale snažil jsem se experimentálně zjistit, co je pro běh jádra a především mého programu v něm ještě třeba a jsou zde tedy části, které jsou buď nezbytně nutné pro běh mého programu nebo také takové, u kterých jsem předpokládal jejich budoucí využití nebo si nebyl přesně jist, zda opravdu nebudou již třeba.

- **Machine selection**

Tato část se zabývá výběrem našeho systému, kde zvolíme:

```
…\Systém type = SGI IP32 (O2)
```

- **Endianess selection**

Zvolíme položku `Big endian`

- **CPU selection**

Zde můžeme vybrat, na jaký druh procesoru se má jádro připravit:

```
…\CPU type = R5000
```

Máme sice procesor R10000, ale toto jádro umí pracovat na systému IP32 jen s procesorem R5000, RM52xx a RM7000. Procesor R5000 je však neblíže podobný našemu použitému a není problém použít právě toto nastavení.

- **Kernel type**

Položky nastavující typ jádra nastavíme následovně:

```
…\Kernel code model = 64-bit kernel
…\Kernel page size = 4kB
…\MIPS MT options = Disable multithreading support
…\Memory model = Flat memory
```

Dále zatrhneme položku:

```
…\Add LRU list to track non-evictable pages
```

A nastavíme:

```
…\Timer frequency = 250 HZ
…\Preemption Model = No Forced Preemption (Server)
```

[12] http://packages.debian.org/lenny/libncurses5-dev

Tímto máme nastaven **Kernel type**, kde by většina těchto hodnot měla být nastavena i už po zadání příkazu **make ip32_defconfig**, zmíněném dříve.

- **General setup**

Zde jsem ponechal většinu položek zaškrtnutých podle výchozího nastavení. Proto zde uvedu tentokrát jen položky, které jsou **NEZAŠKTNUTY**:

```
…\Local version – append to kernel release = hodnota nenastavena
…\BSD Process Accounting vision 3 file format
…\Control Group support
…\Group CPU scheduler
…\Namespaces support
…\Initial RAM filesystem and RAM disk (initramfs/inird) support
…\Profiling support (EXPERIMENTAL)
…\Activate markers
…\Configure standard kernel feature (for small system)\Do an extra kallsyms pass
```

Jen je třeba ještě zkontrolovat nastavení následující položky:

```
…\Choose SLAB allocator = SLAB
```

- **Enable loadable module support**

Celé odškrtneme.

- **Enable the block layer**

V hlavním menu ponecháme zaškrtnuto, ale všechny podpoložky odškrtneme, kromě následujících:

```
…\IO scheduler\Anticipatory I/O scheduler
…\IO scheduler\Deadline I/O scheduler
…\IO scheduler\CFQ I/O scheduler
…\IO scheduler\Default I/O scheduler = CFQ
```

- **Bus option (PCI, PCMCIA, EISA, ISA, TC)**

Zaškrtneme jen položku:

```
…\Support for PCI controller
```

- **Executable file formats**

Zaškrtneme následující položky:

```
…\Kernel support for ELF binaries
…\Kernel support for MISC binaries
…\Kernel support for Linux\MIPS 32-bit binary compatibility
```

- **Power management option**

Celé odškrtneme.

- **Networking support**

Celé odškrtneme.

- **Device Drivers**

Zaškrtneme všechny položky v Generic Driver Options a nastavíme následující:

```
…\Generic Driver Options\path to euvent helper = /sbin/hotplug
…\Generic Driver Options\External firmware blobs to build into the kernel binary = žádná hodnota
```

Ještě nastavíme Input device support a Character devices:

```
…\Input device support\Generic input layer (needed for keyboard, mouse, …)
…\Input device support\Event interface
…\Input device support\Keyboards\AT keyboard
…\Input device support\Hardware I/O ports\Serial I/O support
…\Input device support\Hardware I/O ports\PCI PS/2 keyboard and PS/2 mouse controller
…\Input device support\Hardware I/O ports\PS/2 driver library
…\Input device support\Hardware I/O ports\Raw access to serio ports
…\Character devices\Virtual terminal
…\Character devices\Enable character translation in console
…\Character devices\Support for console on virtual terminal
…\Character devices\/dev/kmem virtual device support
…\Character devices\Unix98 PTY support
…\Character devices\Hardware Random Number Generator Core support
```

Důležité je především nastavení v Graphics support, jelikož právě to nám umožňuje používat výstup na obrazovku.

```
…\Graphics support\Lowlevel video output switch controls
…\Graphics support\Support for frame buffer devices\Enable firmware EDID
…\Graphics support\Support for frame buffer devices\SGI Graphics Backend frame buffer support
…\Graphics support\Support for frame buffer devices\Video memory size in M=1
…\Graphics support\Console display driver support\VGA text console
…\Graphics support\Console display driver support\Framebuffer Console support
```

Zde je také nastavení velikosti grafické paměti, která má standardně velikost 4MB a její velikost se určuje v jednotkách MB, avšak pro korektní chod framebufferu je třeba, aby její velikost byla minimálně 1MB. Další důležitou položkou pro běh framebufferu, je také položka Framebuffer Console support.

- **File systems**

Zde zůstanou zaškrtnuty všechny položky Pseudo filesystems a v Native language support jsem zvolil:

```
…\Native language support\Default NLS support = iso8859-1
…\Native language support\Codepage 850(Europe)
…\Native language support\ASCII (United States)
…\Native language support\NLS ISO 8859-1 (Latin 1; Western Europe Language)
```

- **Kernel hacking**

Zaškrtnuty jsem ponechal položky:

```
…\Enable __deprecated logic
…\Enable __must_check logic
…\Warn for stack frames larger than (need gcc 4.4) = 2048
…\Magic SysRq key
…\Sysctl checks
```

- **Security options**

Zde zůstalo zašktnuto jen:

```
…\Enable access key retention support
…\Enable the /proc/keys file by whitch keys may be viewed
```

- **Cryptographic API**

Celé odškrtneme.

- **Library routines**

Zde zůstaly zachovány položky:

```
…\CRC16 function
…\CRC calculation for the T10 Data Integrity Fiels
…\CRC ITU-T V.41 functions
…\CRC32 function
…\CRC32c (Castagnoli, et al) Cyclic Redundancy-Check
```

Můj konfigurační soubor

Další podrobnosti a nastavení konfigurace pro moji stanici se nacházejí na přiloženém CD-ROM disku v souboru o2_final.cfg

6.2.4 Aplikace testující paměť v jádře Linuxu

Jádro Linuxu je psáno jak v jazyce asembler (např. inicializace jádra) a také v jazyce C.

V mojí práci jsem se pokusil nalézt místo v jádře Linuxu, kde by bylo možno spustit můj kód s testem paměti. Cesta k tomuto místu však nebyla jednoduchá a zpočátku se podobala střílení od pasu a hledání čehokoliv co se objevilo na obrazovce ve zdrojovém kódu jádra. Prvním záchytným bodem se staly chybové výpisy při inicializaci kořenového adresáře („*VFS: Cannot open root device …*“), které se nacházejí v souboru **do_mount.c** ve funkci **mount_block_root**. Zde jsem byl také poprvé schopen spustit můj test paměti, který mě posloužil pro vyzkoušení si jednotlivých možností v jádře linuxu.

Další moje práce se zaměřila na posunutí mého testu paměti někam blíže začátku spouštění jádra. A tak jsem začal, vyhledával jsem odkud se moje funkce volá a tímto jsem se přes funkce **mount_root** a **prepare_namespace** dostal až na funkce v souboru **main.c**, **kernel_init**, **rest_init** a **start_kernel**, která je již volána přímo z počátečních částí jádra psaných v asembleru.

Nakonec se ukázala jako nejpoužitelnější volba v souboru **main.c**[13] právě funkce **kernel_init**, kde je již vhodně inicializován framebuffer, ale není ještě příliš mnoho součástí jádra v paměti.

Moje aplikace obsahuje pokusné testy z předchozího programu Memtester, jen musely být upraveny všechny funkce pro výpisy, jelikož v jádře je místo funkce **printf()** funkce **printk()**. Aplikace také

[13] /data/jadro/linux-2.6.28.4/init/main.c

nedisponuje uživatelským menu pro výběr testů a slouží tak spíše jako demonstrace funkcionality testování v jádře Linuxu.

Jelikož se mě nepodařilo rozumně vyřešit odchytávání chyb při alokaci paměti, obsahuje má aplikace 2 režimy, kde v prvním, detekčním režimu, proměnná **detect_mode=1**, dokáže zjistit velikost dostupné paměti, což provádí tak, že se snaží cyklicky alokovat paměť vždy o 1MB větší a až narazí na limit dostupné paměti a zobrazí hlášku, že došla veškerá volná paměť. V druhém, testovacím režimu, proměnná **detect_mode=0**, provede test paměti definované proměnnou **mem_size**.

V mé práci se mi podařilo s již zmíněným konfiguračním souborem dosáhnout, při 256MB RAM, testovatelné velikosti paměti 244MB. Tato velikost je však jen experimentálně změřena a jedná se o hodnotu o 1MB menší než poslední hodnota zobrazená v detekčním režimu.

7 Závěr

Moje práce se zabývá zprovozněním a vytvořením vlastních testů paměti na grafických stanicích SGI O2, což sebou přineslo objevování architektury MIPS a možnost vytváření programů pro systémy založené na jiných architekturách pomocí křížových kompilátorů.

Praktická část práce se dá rozdělit do dvou částí, kde v první jsem si vytvořil v jazyce C psaný program a spustitelný přes síť v prostředí PROM, který svou lepší uživatelskou přívětivostí a především možnostmi, které prostředí PROM poskytuje, sloužil jako základní aplikace pro testování jednotlivých součástí programu a testovacích algoritmů. Tento program je však omezen prostředím PROM a z paměti si dokáže ukousnout, při 256MB instalované paměti, jen přibližně 235 MB.

Ve druhé části práce jsem se zaměřil na možnost získání více volné paměti pro testování, kde se již bylo třeba oprostit od výhod poskytnutých prostředím PROM, a má práce se zaměřila na experimentování s jádrem Linuxu. Zde bylo třeba zpočátku vyřešit problém se zavedením a spuštěním jádra Linuxu, což se podařilo pomocí programu tip22. Následovala fáze hledání vhodného místa pro vložení funkce s algoritmy pro testování paměti, které bylo nalezeno ve funkci kernel_init. Po nalezení vhodného místa pro spouštění testovací funkce, jsem se zaměřil na optimalizaci velikosti jádra, což mělo výrazný vliv na konečnou velikost testované paměti, která dosáhla velikosti až 244 MB, při 256MB instalované paměti a konfiguračním souboru o2_final.cfg.

Avšak ani tato hodnota nemusí být konečná a další vývoj v této oblasti by mohl směřovat, jednak k ještě další optimalizaci a také směrem ještě blíže na začátek samotné inicializace a startu jádra.

Dalším směrem by mohlo být také použití již získaných znalostí k práci i na dalších architekturách a stanicích od SGI, kde program pro prostředí PROM by měl být přeložitelný a použitelný kromě SGI O2 také na stanicích SGI Indy a SGI Octane, obdobně jako druhá část, která by měla být spustitelná i na řadě dalších architektur.

Zajímavou kapitolou je právě již zmíněná možnost práce na jiných architekturách, především proto, že SGI již přestala používat procesory architektury MIPS ve svých stanicích a nahradila je procesory architektury IA64 a X86. Avšak získané zkušenosti především s křížovým překladačem je možno použít i v dalších oblastech například i pro tvorku programů pro nově se prosazující architektury ARM apod.

Z pohledu testování paměti byly testy ověřeny vzhledem k vestavěnému testu paměti v systému IRIX. Při testování, konkrétně při výměně testovaných paměťových bloků, došlo bohužel také k nepříjemné události, kde z neznámých příčin byla poškozena testovaná stanice LAMBDA, což jen dokládá již velmi velkou citlivost těchto systémů.

Moje práce dále poslouží k testování paměťových modulů v grafických stanicích SGI ve společnosti Evektor spol. s r.o., od které jsem také získal testované stanice.

8 Literatura

[1] **Silicon Graphics, Inc:** Unified Memory Architecture. [Online] 1997. [Citace: 6. Květen 2009.] Dostupné na URL: <http://www.futuretech.blinkenlights.nl/o2/1352.pdf>

[2] **http://en.wikipedia.org:** SGI O2. *http://en.wikipedia.org.* [Online] 19. Duben 2009. [Citace: 3. Květen 2009.] Dostupné na URL: <http://en.wikipedia.org/wiki/SGI_O2>

[3] **jirka:** Chvalozpěv na SGI O2. *Penguin.* [Online] 14. Červenec 2005. [Citace: 5. květen 2009.] Dostupné na URL: <http://www.penguin.cz/novinky-view.php?id=1066>

[4] **MIPS Technologies, Inc.:** MIPS R10000 Microprocessor. [Online] 1996. [Citace: 4. Leden 2009.] Dostupné na URL: <http://techpubs.sgi.com/library/manuals/2000/007-2490-001/pdf/007-2490-001.pdf>

[5] **Demeulemeester, Samuel:** *Memtest86+ Advanced Memory Diagnostic Tools.* [Online] 22. Prosinec 2008. [Citace: 5. Leden 2009.] Dostupné na URL: <http://www.memtest.org/#downcode>

[6] **Sevy, Jonathan:** MIPS Architecture and Assembly Language. *MIPS Quick Reference.* [Online] [Citace: 7. Leden 2009.] Dostupné na URL: <http://gicl.cs.drexel.edu/people/sevy/architecture/MIPSRef(SPIM).html>

[7] **debian:** 4.3. Příprava souborů pro zavedení ze sítě pomocí TFTP. *debian.org.* [Online] 28. Duben 2008. [Citace: 3. Leden 2009.] Dostupné na URL: <http://www.debian.org/releases/stable/mips/ch04s03.html.cs#dhcpd>

[8] **Michlmayer, Martin:** Debian on SGI O2. *Cyrius.* [Online] 15. září 2008. [Citace: 3. leden 2009.] Dostupné na URL: <http://www.cyrius.com/debian/o2/>

[9] **Harrison, Peter:** Telnet, TFTP and XINETD. *www.chinalinuxpub.com.* [Online] www.linuxhomenetworking.com, 2. Únor 2008. [Citace: 26. Březen 2009.] Dostupné na URL: <http://www.chinalinuxpub.com/doc/www.siliconvalleyccie.com/linux-hn/xinetd.htm>

[10] **http://cs.wikipedia.org:** Křížová_kompilace. *http://cs.wikipedia.org.* [Online] 3. květen 2009. [Citace: 11. květen 2009.] Dostupné na URL: <http://cs.wikipedia.org/wiki/Křížová_kompilace>

[11] **LinuxMIPS:** ARCLoad. *linux-mips.org.* [Online] 18. Březen 2009. [Citace: 24. Březen 2009.] <http://www.linux-mips.org/wiki/ARCLoad>

[12] **Total Knowledge:** SGI O2 BootCD HOW-TO. *Total Knowledge.* [Online] 3. únor 2009. [Citace: 20. duben 2009.] Dostupné na URL: <http://www.total-knowledge.com/progs/mips/SGI-BootCD-HOWTO.shtml>

[13] **debian:** DebianInstaller. *wiki.debian.org.* [Online] 5. Duben 2009. [Citace: 13. Duben 2009.] Dostupné na URL: <http://wiki.debian.org/DebianInstaller>

[14] **debian:** Details of packege tip22. *debian.org.* [Online] 19. Duben 2009. [Citace: 20. Duben 2009.] Dostupné na URL: <http://packages.debian.org/lenny/tip22>

[15] **SGI, Inc.:** vh(7M). *SGI Techpubs library.* [Online] SGI, Inc., 1993-2007. [Citace: 13. 5 2009.] Dostupné na URL: <http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650&db=man&fname=/usr/share/catman/a_man/cat7/vh.z>

9 Seznam příloh:

9.1 CDROM

Obsahující:

- Text práce
- Zdrojové texty
- Manuál programu Memtester
- Konfigurační soubory
- Použité balíčky knihoven